**PROJET IA**

Notre projet se découpe en plusieurs parties majeures.

Nous commençons tout d’abord par demander à l’utilisateur le nombre de villes qu’il souhaite avoir. Nous vérifions que le texte entré est bel et bien un nombre compris entre 1 et 50. Nous générons ensuite dix circuits aléatoires avec le nombre de villes précédemment indiqué. Ces circuits constituent notre population initiale. La distance entre chaque couple de villes est affichée dans une matrice. La distance totale pour chacun des dix parcours est également affichée.

Nous choisissons les cinq meilleurs circuits dans notre population grâce au fitness. Pour cela, nous divisons 1 par la distance totale d’un circuit, plus le résultat est proche de 1, plus la distance sera faible et donc plus notre circuit sera intéressant. Nous devons ensuite créer notre nouvelle population.

L’algorithme génétique utilise les croisements et mutations pour la création de nouveaux individus, nous faisons de même avec notre population. Pour simuler une mutation, nous avons choisi d’interchanger deux villes d’un circuit de façon aléatoire. Les croisements sont, quant à eux, effectués en prenant deux circuits distincts, chaque circuit est « coupé » en deux, la première partie du circuit1 et la seconde partie du circuit2 forment le nouvel individu1, la première partie du circuit2 et la seconde partie du circuit1 forment le nouvel individu2. Nous obtenons ainsi une nouvelle génération de dix individus, constituée des cinq meilleurs de notre première génération et des cinq nouveaux résultant des mutations et croisements. Nous répétons l’étape plusieurs fois afin d’avoir une population de plus en plus intéressante, c’est-à-dire des circuits avec des valeurs de plus en plus faibles.

Au bout de la i-ème itération, nous obtenons notre population finale. Nous sélectionnons le circuit dont la distance est la plus faible et nous affichons le graphe qui en résulte.

ANNEXE

import tkinter

from tkinter import \*

import random

*# Listes des noms de villes*

ListeVilles = ["Évry","Marseille","Lyon","Toulouse","Nice","Nantes","Strasbourg","Montpellier","Bordeaux","Lille",

               "Rennes","Reims","Le Havre","Saint-Étienne","Toulon","Grenoble","Angers","Dijon","Brest","Le Mans",

               "Nîmes","Aix-en-Provence","Clermont-Ferrand","Tours","Amiens","Limoges","Villeurbanne","Metz",

               "Besançon","Perpignan","Orléans","Caen","Mulhouse","Boulogne-Billancourt","Rouen","Nancy",

               "Argenteuil","Montreuil","Saint-Denis","Roubaix","Paris","Maisons-Alfort","Cergy","Laval",

               "Cannes","Saint-Hilaire","Viry-Châtillon","Cesson","La Rochelle","Melun","Hyères"]

*##############################################################################*

**def** entryValid():

    var = False

    print("\n")

    if len(nbVilles.get())>0 and len(nbVilles.get())<=2 and (nbVilles.get())!="0" and (nbVilles.get()).isdigit():

        if int(nbVilles.get())<=50:

            print(**f**"Vous avez choisi {nbVilles.get()} villes")

            var = True

    if var == False:

        print("Le nombre de villes doit être compris entre 1 et 50")

        bouton.config(state = tkinter.DISABLED)

    else:

        bouton.config(state = tkinter.ACTIVE)

    return var

*##############################################################################*

*# Fonction pour générer un nom de ville aléatoire*

**def** genererNomVille():

*# Si la liste est vide, on affiche une erreur*

    if not ListeVilles:

        raise ValueError("Le nombre maximal de ville est de 50.")

*# On choisit aléatoirement la ville parmi la liste de villes*

    Villes = random.choice(ListeVilles)

*# On supprime la ville choisie afin d'éviter de réutiliser la ville plusieurs fois*

    ListeVilles.remove(Villes)

    return Villes

*##############################################################################*

*# Fonction pour générer une matrice de distances aléatoires entre les villes*

**def** genererMatriceDistances(n):

    matrice = [[0] \* n for \_ in range(n)]

    for i in range(n):

        for j in range(i + 1, n):

            distance = random.randint(1, 100)

            matrice[i][j] = distance

            matrice[j][i] = distance

    return matrice

*##############################################################################*

*# Fonction pour générer une liste de n villes aléatoires*

**def** genererVilles(n):

    villes = []

    suppVilles = []

    for \_ in range(n):

        nom = genererNomVille()

        villes.append(nom)

        suppVilles.append(nom)

    for v in suppVilles:

        ListeVilles.append(v)

    return villes

*##############################################################################*

*# Fonction pour calculer la distance totale d'un circuit*

**def** distance\_totale(circuit, matrice\_distances):

    distance\_totale = 0  *# Réinitialiser la distance à 0 pour chaque circuit*

    for i in range(len(circuit)):

        ville1 = circuit[i]

        if i == len(circuit) - 1:

            ville2 = circuit[0]  *# Retour à la ville de départ*

        else:

            ville2 = circuit[i + 1]

        distance\_totale += matrice\_distances[ville1][ville2]

    return distance\_totale

*##############################################################################*

*# Fonction pour afficher une matrice carrée*

**def** afficherMatriceCarree(matrice):

    n = len(matrice)

    max\_len = max(len(str(matrice[i][j])) for i in range(n) for j in range(n))

    format\_str = "{:" + str(max\_len) + "}"

    for i in range(n):

        ligne = [format\_str.format(matrice[i][j]) for j in range(n)]

        print(" ".join(ligne))

*##############################################################################*

*# Fonction pour générer une population initiale de circuits aléatoires*

**def** generer\_population\_initiale(n, taille\_population):

    population = []

    for i in range(taille\_population):

        circuit = list(range(n))

        random.shuffle(circuit)

        population.append(circuit)

    return population

*##############################################################################*

*#fonction pour échanger le début et la fin de deux circuits*

**def** croisement(circuit1,circuit2):

    deb = 0

    fin = len(circuit1)//2

    newCircuit1 = []

    newCircuit2 = []

    for i in range(deb,fin):

            newCircuit1.append(circuit1[i])

            newCircuit2.append(circuit2[i])

    for i in range(fin, len(circuit1)):

        if circuit2[i] not in circuit1:

            newCircuit1.append(circuit2[i])

        if circuit1[i] not in circuit2:

            newCircuit2.append(circuit1[i])

    print(**f**"{newCircuit1}")

    print(**f**"{newCircuit2}")

*##############################################################################*

*# Fonction pour calculer le fitness*

**def** calculer\_fitness(circuit, matrice\_distances):

    distance\_totale = distance\_totale(circuit, matrice\_distances)

    if distance\_totale == 0:

        return float('inf')  *# Éviter la division par 0*

    else:

        return 1 / distance\_totale

*##############################################################################*

*# Fonction qui selectionne les 5 meilleurs circuits en fonction du fitness*

**def** selection\_par\_tri(population, fitness\_population):

*# Trier la population par ordre croissant de fitness*

    population\_triee = sorted(zip(population, fitness\_population))

*# Sélectionner les 5 meilleurs circuits*

    parents = [circuit for circuit, \_ in population\_triee[:5]]

    return parents

*##############################################################################*

**def** nouvelle\_population():

    print("")

*##############################################################################*

*# Fonction pour muter un circuit*

**def** mutation(circuit):

*# On choisit 2 villes au hasard*

    ville1, ville2 = random.sample(range(len(circuit)), 2)

*# On inverse leur ordre*

    circuit[ville1], circuit[ville2] = circuit[ville2], circuit[ville1]

    return circuit

*##############################################################################*

**def** graphe():

    bouton.config(state = tkinter.DISABLED) *# -> bouton grisé = bouton non cliquable*

    n = int(nbVilles.get())

    villes = genererVilles(n)

    matrice\_distances = genererMatriceDistances(n)

    circuit = list(range(n))

    random.shuffle(circuit)

    taille\_population = 10 *# nombre de circuit*

    distance = distance\_totale(circuit, matrice\_distances)

    print("--------------------------------------------------------------------------------------")

    print("Distance totale du circuit initial :", distance)

    print("Circuit :", [villes[i] for i in circuit])

    print("\n")

    afficherMatriceCarree(matrice\_distances)

    print("\n")

    population = generer\_population\_initiale(n, taille\_population)

    for i, circuit in enumerate(population):

        distance = distance\_totale(circuit, matrice\_distances)

        print(**f**"Circuit {i+1} (distance = {distance}) : {[villes[j] for j in circuit]}")

*############################### AFFICHAGE ###############################*

fenetre = tkinter.Tk()

fenetre.title("Affichage du graphe")

fenetre.minsize(width=216, height=47)

fenetre.configure(pady=10)

txt = tkinter.Label(text = "Nombre de villes :")

txt.grid(column = 0, row = 0)

nbVilles = tkinter.Entry(justify=CENTER, width=5)

nbVilles.grid(column = 1, row = 0)

bouton = tkinter.Button(text = "GO!", activebackground="green", activeforeground="white")

bouton.grid(column = 4, row = 0)

bouton.config(command = graphe) *# pr lancer notre fonction quand le button est pressé*

valid = tkinter.Button(text = "Valider", activebackground="green", activeforeground="white")

valid.grid(column = 3, row = 0)

valid.config(command = entryValid)

fenetre.mainloop()